Методичка 2.4 - Работа с функциями

Создание функции

Для того, чтобы создать функцию на языке Ассемблер необходимо воспользоваться директивами PROC - это начало функции и ENDP - конец функции. Перед директивой PROC указывается имя функции.

Также при создании функции стоит придерживаться соглашения, что значение, которое возвращает функция, должно храниться в регистре EAX. Это не является обязательным требованием, но если кто-то будет использовать вашу функцию в своем коде, то он будет ожидать, что результат будет в регистре EAX.

Исходный код программа prog-1.4.1.asm

```
.386
.model flat, stdcall

_text segment
start:
xor eax, eax
call my_func
add eax, 4
ret

my_func proc
xor eax, eax
inc eax
ret
my_func endp

_text ends
end start
```

В данной программе присутствует функция с именем my_func. Вызов функции осуществляется с помощью оператора CALL. Оператор CALL не только осуществляет переход к функции, но и сохраняет адрес следующей инструкции после себя в стеке. Это делается для того, чтобы программа понимала куда нужно вернуться после того, как функция будет выполнена.

Возврат из функции осуществляется с помощью оператора RET. Оператор RET забирает сохраненный адрес с вершины стека и переходит по нему.

Компиляция:

> ml /c /coff prog-1.4.1.asm
> link /SUBSYSTEM:CONSOLE /DYNAMICBASE:NO prog-1.4.1.obj

Получаем файл prog-1.4.1.exe.

Запускаем отладчик и открываем программу prog-1.4.1.exe.

В дизассемблированном листинге уже нет названия функции после оператора CALL, а есть только адрес, который указывает на начало функции.

Обратите внимание, что оператор CALL выполняет вызов функции по адресу 0x00000004. Дело в том, что операторы CALL бывают разные. В данном случае используется оператор CALL, который вызывает функцию по относительному адресу, а не по абсолютному. Оператор CALL, который вызывает функцию по относительному адресу имеет опкод - E8, а для вызова функции по абсолютному адресу используется оператор CALL с опкодом - 9A. Вызов функции по относительному адресу называется - Near Call, а по абсолютному - Far Call. В программах для ОС Windows, почти всегда, используется Near Call.

Перед тем, как выполнить инструкцию CALL необходимо запомнить, что находится на вершине стека. Также нужно запомнить адрес следующей инструкции после CALL. Затем происходит вызов функции. Необходимо использовать команду Step Into (F7), чтобы попасть внутрь функции.

Переходим внутрь функции и на вершине стека появляется новое значение, которое равно адресу следующей инструкции после инструкции CALL.

После выполнения всех инструкций функции необходимо выполнить остановку на инструкции RET. Необходимо заполнить значение на вершине стека и выполнить инструкцию. Происходит переход по адресу, который был на вершине стека и этого адреса уже в стеке нет.

Принцип работы стека

Ранее было сказано, что оператор CALL сохраняет адрес следующей инструкции в стеке, чтобы знать куда нужно вернуться после выполнения функции.

Когда происходит запуск программы, в операционной системе создается новый процесс. В операционной системе Windows каждый процесс имеет свое собственное адресное пространство. В этом адресном пространстве имеется несколько областей:

Исполняемый файл (.exe)
Динамические библиотеки (dll)
Куча (heap)
Стек (stack)

Стек растет от старших адресов к младшим.

Можно сказать, что стек является областью памяти для хранения временных данных. Например, стек используется для передачи аргументов в функцию или для сохранения адреса возврата, как это делает оператор CALL. В процессе работы программы стек то увеличивается, то уменьшается.

Для работы со стеком в языке Ассемблер существуют специальные операторы: PUSH и POP, которые позволяют класть значения в стек или наоборот забирать из него. Пример использования данных операторов:

```
push eax ;  положить значение в регистре EAX на вершину стека
push edx ;  положить значение в регистре EDX на вершину стека
pop ecx ; забрать значение с вершины стека и положить его в регистр ECX
```

Также стоит вспомнить, что процессор имеет специальные регистры по работе со стеком. Это регистры ESP и EBP, первый всегда указывает на вершину стека, второй на начало стекового фрейма. Границы стекового фрейма необходимы для разграничения локальных переменных разных функций.

Исходный код программы prog-1.4.2.asm

```
.386
.model flat, stdcall

_text segment
start:
xor eax, eax
inc eax
push eax

mov ebx, 4
push ebx

pop ecx
pop edx

ret
_text ends
end start
```

В данной программе сначала происходит установка определенного значения в регистре EAX и затем оно попадает на вершину стека, затем происходит работа с регистром EBX и также оно

попадает на вершину стека. Затем с помощью оператора POP дважды забирается значение из стека, сначала в регистр ECX, а потом в EDX.

Компиляция:

```
> ml /c /coff prog-1.4.2.asm
> link /SUBSYSTEM:CONSOLE /DYNAMICBASE:NO prog-1.4.2.obj
```

Получаем файл prog-1.4.2.exe.

Запускаем отладчик и открываем программу prog-1.4.2.

Пошагово выполняем программу (F8).

Функции WinAPI

Функции WinAPI созданы для того, чтобы разработчикам было удобно взаимодействовать с операционной системой Windows. Можно сказать, что это системные функции Windows. Например, функция MessageBox, которая просто создает окно с надписью является функцией WinAPI.

Если в процессе разработки программы просто указать название данной функции, то при линковке будет ошибка, так как линкер не будет знать, где находится реализация указанной функции. Линкеру нужно указать библиотеку, в которой есть реализация этой функции. В случае с функцией MessageBox, реализация данной функции расположена в библиотеке user32.lib.

Узнать в какой библиотеке реализована та или иная функция можно из документации. Документация доступна на официальном сайте компании Microsoft. Также можно поискать в Google.

Исходный код программы prog-1.4.3.asm

```
.386
.model flat, stdcall

ExitProcess PROTO, ExitCode:DWORD
MessageBoxA PROTO, hWnd:DWORD, lpText:DWORD, lpCaption:DWORD, uType:DWORD

_data segment
messageText db 'Some Text',0
messageTitle db 'Some Title',0
_data ends

_text segment
```

```
start:
push 0
push offset messageTitle
push offset messageText
push 0
call MessageBoxA

push 0
call ExitProcess
_text ends
end start
```

Программа использует 2 функции WinAPI: это MessageBox и ExitProcess. По хорошему, каждая программа в Windows должна завершаться функцией ExitProcess.

В данном случае программа создает окно с помощью функции MessageBox и завершается с помощью функции ExitProcess.

Прежде чем вызвать функцию, надо узнать какие входные аргументы требует функция, в этом обычно помогает документация.

В самом начале программы необходимо написать прототипы функций. Название аргумента и через двоеточие размер этого аргумента. Название аргумента пишется для удобства программиста, чтобы не запутаться, но они не будут использоваться при компиляции. А вот размер имеет значение, в данном случае все аргументы имеют размер 32 бита, т.е. DWORD.

В секции данных есть две строки: заголовок окна и основной текст в окне.

Далее идет секция кода. Важно в правильном порядке класть входные аргументы для функции в стек и затем происходит вызов функции. В данном случае используется 4 аргумента для функции MessageBox и 1 аргумент для функции ExitProcess.

Компиляция.

```
> ml /c /coff prog-1.4.3.asm
> link /SUBSYSTEM:WINDOWS /DYNAMICBASE:NO prog-1.4.3.obj kernel32.lib user32.lib
```

Получаем файл prog-1.4.3.exe.

Посредством параметра SUBSYSTEM можно указать значение - WINDOWS, что говорит о том, что приложение имеет графический интерфейс и его не следует запускать через командную строку.

Заметьте, что при линковке мы указываем не только наш объектный файл, но и библиотеки, которые необходимы.

Запускаем отладчик и открываем программу prog-1.4.3.exe.

В самом начале расположены 4 инструкции PUSH, которые подготавливают входные параметры для функции MessageBoxA.

Далее происходит вызов функции с помощью оператора CALL.

```
...
0040100E      E8 0D000000          CALL <JMP.&USER32.MessageBoxA>
...
```

Вызов функции MessageBoxA происходит по относительному адресу - 0D000000.

Ставим поставим точку останова после инструкции CALL (F2). Затем необходимо выполнить вызов функции (F7) и тогда происходит переход на инструкцию JMP.

```
00401020      FF25 08204000        JMP DWORD PTR DS:[<&USER32.MessageBoxA>]
```

В инструкции JMP адрес 0x00402008 (стоит помнить про little endian). Этот адрес указывает на таблицу импорта, которая содержит абсолютные адреса импортируемых функций.

Переходим по этому адресу (Ctrl - G, 0x00402008).

Там располагается адрес 0x76367E90. Это и есть абсолютный адрес функции MessageBoxA в библиотеке user32.dll.

Адрес от запуска к запуску может отличаться, так как динамическая библиотека user32.dll скомпилирована с поддержкой ASLR.

Затем необходимо выполнить инструкцию JMP и происходит переход именно по этому адресу.

Затем продолжаем выполнение программы (F9) и возвращаемся к коду.

Вызов функции ExitProcess происходит аналогично.